INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

The game of the name in cryptographic tables

Roberto M. Amadio Sanjiva Prasad

N° 3733

Juillet 1999

THÈME 1

ISSN 0249-6399

The game of the name in cryptographic tables

Roberto M. Amadio Sanjiva Prasad

Thème 1 — Réseaux et systèmes
Projet Meije

Rapport de recherche n° 3733 — Juillet 1999 — 17 pages

Abstract: We present a name-passing calculus that can be regarded as a simplified π-calculus equipped with a *cryptographic table*. The latter is a data structure representing the relationships among names. We apply the calculus to the modelling and verification of secrecy and authenticity properties in cryptographic protocols relying on symmetric shared keys. Following classical approaches [8], we formulate the verification task as a reachability problem and prove its decidability assuming finite principals and bounds on the sorts of the messages synthesized by the attacker.

Key-words: Cryptographic protocols, π-calculus, verification

The first author works at *Université de Provence*, Marseille. and he is an external collaborator of the INRIA-Ecole des Mines Project *Meije*. He can be contacted at the following address: CMI, 39 rue Joliot-Curie, F-13453, Marseille, France. `mailto: amadio@gyptis.univ-mrs.fr`. He work was partly supported by *Action Coopérative PRESYSA*, *IFCPAR 1502-1*, LIM (CNRS ES-A6077), and *WG Confer*. The second author works at the Indian Institute of Technology, Delhi. `mailto: sanjiva@cse.iitd.ernet.in`. He was partly supported by *IFCPAR 1502-1* and *AICTE 1-52/CD/CA(08)/96-97* and BRICS, Centre of the Danish National Research Foundation.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, B.P. 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : 04 92 38 77 77 - International : +33 4 92 38 77 77 — Fax : 04 92 38 77 65 - International : +33 4 92 38 77 65

Le jeu du nom dans les tables cryptographiques

Résumé : Nous présentons un calcul avec passage de noms qui peut être consideré comme un π-calcul simplifié equipé avec une *table cryptographique*. Il s'agit d'une structure de données qui représente les relations entre les noms. Nous appliquons le calcul à la modelisation et vérification de proprietés de securité et authenticité dans les protocoles cryptographiques à clef symétrique. Suivant des approches classiques [8], nous formulons le problème de vérification comme une proprieté d'accessibilité et nous prouvons sa decidabilité en supposant un nombre fini de principaux et une borne sur les sortes des messages synthétisés par l'attaquant.

Mots-clés : Protocoles cryptographiques, π-calcul, vérification.

1 Introduction

Cryptographic protocols are commonly used to establish secure communication channels between distributed *principals*. Cryptographic protocols seem good candidates for formal verification and several frameworks have been proposed for making possible formal and automatable analyses. We do not attempt to survey here all the various approaches, but confine our remarks to some which assume 'perfect encryption' and model a protocol as a collection of interacting processes competing against a hostile environment.

One class of approaches involves state exploration using model-checking techniques [11, 6, 13]. Lowe [11], Schneider [15] and several others have used CSP to specify authentication protocols, analysing them with the FDR model-checking tool. Other state exploration approaches are based on logic programming techniques [12]. The main benefit of these approaches is their automation and efficacy in uncovering subtle bugs in protocols (*e.g.,* Lowe's 'man-in-the-middle' attack on the Needham-Schroeder symmetric key protocol). However, their applicability is limited by the quickly growing size of the state space to be examined; simplifying hypotheses on the behaviour of the environment necessary to bound the state space may drastically curtail the ability to find errors.

A second class of approaches relies on general-purpose proof assistant tools. Paulson [14] uses induction on traces to formally prove protocol correctness using Isabelle. Bolignano [4] uses a state-based analysis of the protocols, proving invariant properties, with the proofs subsequently mechanized in Coq. Although these approaches are not automatic, recent work [10, 16] suggests that certain authentication protocols can be modelled in decidable fragments of first-order logic.

In all the approaches mentioned above the attacker must be explicitly modelled. A more recent trend has been the use of name-passing process calculi for studying cryptographic authentication protocols. Abadi and Gordon have presented the *spi*-calculus [1], an extension of the π-calculus with cryptographic primitives (see [7] for a related approach using a 'second-order' calculus). Principals of a protocol are expressed in a π-calculus-like notation, whereas the attacker is represented implicitly by the process calculus notion of 'environment'. Security properties are modelled in terms of contextual equivalences, in contrast to the previous approaches. The *spi*-calculus provides a precise notation with a formal operational semantics, particularly for expressing the generation of fresh names, for the scope of names, and for identifying the different threads in a protocol and the order of events in each thread. These features are important: in the various notations found in the literature, the issues of name generation, scoping, data sorts, and synthesis capabilities of the adversary were often treated in an *ad hoc* and/or approximate manner.

Unfortunately, the addition of the cryptographic primitives to the π-calculus considerably complicates reasoning about the behaviour of processes. Although there have been some attempts to simplify this reasoning (see [2, 9, 5]), the developed theory has not yet led to automatic or semi-automatic verification methods.

In this paper, we describe an approach that combines the analyses found in the model-checking and theorem-proving approaches with the benefits of using a process notation. Focussing on symmetric shared-key cryptosystems, we propose a name-passing calculus based

on a variety of π-calculus and use it for modelling and verifying secrecy and authenticity properties in cryptographic protocols. Our approach departs from that of Abadi and Gordon in three major ways.

First, we insist on considering every transmissible value as a name, thus eliminating complications to the theory arising from structured values. We keep track of the relationships amongst names (what name is a ciphertext of what plaintext) by means of a *cryptographic table*. Secondly, we model secrecy and authenticity properties as reachability properties that are largely insensitive to the ordering of the actions and to their branching structure. Intuitively, we designate configurations reached after a successful attack as *erroneous*. Protocol verification then involves showing invariance of the property that such error configurations are not reachable. Thirdly, we eliminate named communication channels and, as in *e.g.*, [4], let all communications between principals and the environment transit on a public medium.

This paper is organized as follows. In section 2, we define a simple name-passing calculus equipped with a cryptographic table, present some derived operators, and relate our formalism to the π-calculus. In section 3, we apply the calculus for modelling a concrete example: the Yahalom protocol. Finally, in section 4 relying on simple rewriting techniques we show that the reachability problem is decidable assuming finite principals and bounds on the sorts of the messages synthesized by the attacker.

2 The calculus

We define a process calculus enriched with a 'cryptographic table' to model and analyse symmetric key cryptographic protocols. We use $a, b, \ldots$ for names and $\mathbf{a}, \mathbf{b}, \ldots$ for vectors of names. N denotes the set of names. The principals' behaviour as well as secrecy and authenticity properties are represented as processes.

Definition 2.1 (processes) *A process (typically p, q) is defined by the following grammar:*

$$\begin{array}{rl} p ::= & \mathbf{0} \parallel err \parallel !a.p \mid ?a.p \mid (\nu a)\, p \mid [a = b]p, q \mid p \mid q \mid A(\mathbf{a}) \\ & \mid \mathsf{let}\ a = \{\mathbf{b}\}_c\ \mathsf{in}\ p \parallel \mathsf{case}\ \{\mathbf{b}\}_c = a\ \mathsf{in}\ p\ . \end{array}$$

As usual, $\mathbf{0}$ is the terminated process; err is a distinguished 'error' process; $!a.p$ sends a to the environment and becomes p; $?a.p$ receives a name from the environment, binds it to a and becomes p; $(\nu a)\, p$ creates a new restricted name a and becomes p; $[a = b]p, q$ tests the equality of a and b and accordingly executes p or q; $p \mid q$ is the parallel composition of p and q; $A(\mathbf{a})$ is a recursively defined process; $\mathsf{let}\ a = \{\mathbf{b}\}_c\ \mathsf{in}\ p$ defines a to be the encryption of $\mathbf{b}$ with key c in p; finally $\mathsf{case}\ \{\mathbf{b}\}_c = a\ \mathsf{in}\ p$ defines $\mathbf{b}$ to be the decryption of a with key c in p. The input and restriction operators act as name binders. Moreover, a is bound in $\mathsf{let}\ a = \{\mathbf{b}\}_c\ \mathsf{in}\ p$ and the names $\mathbf{b}$ are bound in $\mathsf{case}\ \{\mathbf{b}\}_c = a\ \mathsf{in}\ p$. We denote with $fn(p)$ the set of names free in p. We assume that for every process identifier $A(\mathbf{a})$ there is a unique recursive equation $A(\mathbf{a}) = p$ such that $fn(p) \subseteq \{\mathbf{a}\}$.

Let T be a relation in $(\bigcup_{k\geq 1} N^k) \times N \times N$. We write $a \in n(T)$ if the name a occurs in a tuple of the relation T. We write $(\mathbf{b}, c, a) \in T$ as $\{\mathbf{b}\}_c = a \in T$. This notation is supposed

to suggest that c is a key, $\mathbf{b}$ is a tuple of plaintext, and a is the corresponding ciphertext. The relation T induces an order $<_T$ on $n(T)$ which we define as the least transitive relation such that: $\{b_1,\ldots,b_n\}_c = a \in T \;\Rightarrow\; b_1,\ldots,b_n,c <_T a$.

Definition 2.2 (cryptographic table) *A cryptographic table T is a relation in $(\bigcup_{k\geq 1} N^k)\times N \times N$ which satisfies the following properties:*

$$
\begin{array}{lll}
T \text{ is finite.} & & \\
<_T \text{ is acyclic.} & & \\
\{\mathbf{b}\}_c = a \in T \text{ and } \{\mathbf{b}\}_c = a' \in T & \Rightarrow a = a' & \textit{(single valued)} \\
\{\mathbf{b}\}_c = a \in T \text{ and } \{\mathbf{b}'\}_{c'} = a \in T & \Rightarrow \mathbf{b} = \mathbf{b}' \text{ and } c = c' & \textit{(injectivity)}\ .
\end{array}
$$

We introduce a notion of *sort* for the names in a cryptographic table.

Definition 2.3 (sorts) *The collection of sorts Srt is the least set that contains the ground sort 0 and such that $(s_1,\ldots,s_n) \in Srt$ if $s_i \in Srt$ for $i = 1,\ldots,n$ with $n \geq 2$.*

Every name occurring in a cryptographic table can be assigned a unique sort as follows.

Definition 2.4 (sorting) *Let T be a cryptographic table. We define a function $srt_T : n(T) \rightarrow Srt$ as follows:*

$$
srt_T(a) = \begin{cases} 0 & \text{if } a \text{ is minimal in } <_T \\ (s_1,\ldots,s_n) & \text{if } \{b_1,\ldots,b_{n-1}\}_{b_n} = a \in T \text{ and } srt_T(b_i) = s_i, i = 1,\ldots,n\ . \end{cases}
$$

Definition 2.5 (configuration) *A configuration r is a triple $(\nu\{\mathbf{a}\})\,(p \mid T)$ where $\{\mathbf{a}\}$ is a* set *of restricted names, p is a process, and T is a cryptographic table.*

We write $r \equiv r'$ if r and r' are identical configurations up to α-renaming of bound names and associativity-commutativity of parallel composition.

In figure 1, we define a *reduction relation* on configurations. The first five rules describe the computation performed by the principals or by 'observer processes' needed in the verification of secrecy or authenticity properties.Rules (out) and (in) concern the sending of a name to the environment and the reception of a name from the environment. Rules (ν), (m), and (rec) describe internal computation performed by the principals: generation of new names, conditional, and unfolding of recursive definitions. The cryptographic table plays a role in the next three rules. In particular, it allows sharing of information between principals and environments. The rules (let^1) and (let^2) compute the ciphertext a' associated with $\{\mathbf{b}\}_c$ in T while adding $\{\mathbf{b}\}_c = a'$ to T if it is not already there. The rule $(case)$ tries to decode the ciphertext a with key c. In the rule $(case)$, a deadlock occurs (specified by the absence of a transition) if either the vectors $\mathbf{b}$ and $\mathbf{b}'$ do not have the same length or an incorrect key is used for decoding.

Finally, the last three rules (let^1_e), (let^2_e), and $(case_e)$ describe the encoding/synthesis and decoding/analysis performed by the environment: in the rule (let^1_e) the environment learns

(out) $\quad (\nu\{\mathbf{a}\})\,(!a.p \mid q \mid T) \to (\nu\{\mathbf{a}\}\backslash a)\,(p \mid q \mid T)$

(in) $\quad (\nu\{\mathbf{a}\})\,(?a.p \mid q \mid T) \to (\nu\{\mathbf{a}\})\,([b/a]p \mid q \mid T) \quad \text{if } b \notin \{\mathbf{a}\}$

(ν) $\quad (\nu\{\mathbf{a}\})\,((\nu a)\,p \mid q \mid T) \to (\nu\{\mathbf{a}\} \cup \{a\})\,(p \mid q \mid T) \quad a \notin fn(q) \cup n(T)$

(m) $\quad (\nu\{\mathbf{a}\})\,([a=b]p_1, p_2 \mid q \mid T) \to \begin{cases} (\nu\{\mathbf{a}\})\,(p_1 \mid q \mid T) & \text{if } a = b \\ (\nu\{\mathbf{a}\})\,(p_2 \mid q \mid T) & \text{if } a \neq b \end{cases}$

(rec) $\quad (\nu\{\mathbf{a}\})\,(A(\mathbf{b}) \mid q \mid T) \to (\nu\{\mathbf{a}\})\,([\mathbf{b}/\mathbf{c}]p \mid q \mid T) \quad \text{if } A(\mathbf{c}) = p$

(let^1) $\quad (\nu\{\mathbf{a}\})\,(\mathsf{let}\ a = \{\mathbf{b}\}_c\ \mathsf{in}\ p \mid q \mid T) \to (\nu\{\mathbf{a}\})\,([a'/a]p \mid q \mid T)$
if $\{\mathbf{b}\}_c = a' \in T$

(let^2) $\quad (\nu\{\mathbf{a}\})\,(\mathsf{let}\ a = \{\mathbf{b}\}_c\ \mathsf{in}\ p \mid q \mid T) \to (\nu\{\mathbf{a}\} \cup \{a'\})\,([a'/a]p \mid q \mid T \cup \{\{\mathbf{b}\}_c = a'\})$
if $\nexists\, a'\ (\{\mathbf{b}\}_c = a' \in T)$ and a' is fresh.

$(case)$ $\quad (\nu\{\mathbf{a}\})\,(\mathsf{case}\ \{\mathbf{b}\}_c = a\ \mathsf{in}\ p \mid q \mid T) \to (\nu\{\mathbf{a}\})\,([\mathbf{b}'/\mathbf{b}]p \mid q \mid T)$
if $\{\mathbf{b}'\}_c = a \in T$

(let^1_e) $\quad (\nu\{\mathbf{a}, a\})\,(p \mid T \cup \{\{\mathbf{b}\}_c = a\}) \to (\nu\{\mathbf{a}\})\,(p \mid T \cup \{\{\mathbf{b}\}_c = a\})$
if $\{\mathbf{b}, c\} \cap \{\mathbf{a}, a\} = \emptyset$ and $a \notin \{\mathbf{a}\}$

(let^2_e) $\quad (\nu\{\mathbf{a}\})\,(p \mid T) \to (\nu\{\mathbf{a}\})\,(p \mid T \cup \{\{\mathbf{b}\}_c = a\})$
if $\{\mathbf{b}, c\} \cap \{\mathbf{a}\} = \emptyset$, $\nexists\, a\ (\{\mathbf{b}\}_c = a \in T)$, and a is fresh

$(case_e)$ $\quad (\nu\{\mathbf{a}\})\,(p \mid T \cup \{\{\mathbf{b}\}_c = a\}) \to (\nu\{\mathbf{a}\}\backslash\{\mathbf{b}\})\,(p \mid T \cup \{\{\mathbf{b}\}_c = a\})$
if $\{a, c\} \cap \{\mathbf{a}\} = \emptyset$ and $\{\mathbf{b}\} \cap \{\mathbf{a}\} \neq \emptyset$

Figure 1: Reduction rules

a private name by encoding, in the rule (let_e^2) the environment creates a new ciphertext, and in the rule $(case_e)$ the environment learns new names by decoding.

We write $r \rightarrow_R r'$ to make explicit the reduction rule R being applied. We note that encoding a new tuple (plaintext,key) generates a new ciphertext. It follows that the acyclicity and injective function properties of the cryptographic table are preserved by reduction.

Lemma 2.6 *The set of configurations is closed under reduction.*

Definition 2.7 (error) *A* configuration with error *is a configuration having the shape:* $(\nu\{\mathbf{a}\})\,(err \mid p \mid T)$.

We write $r \downarrow err$ if r is a configuration with error (read r commits on err) and $r \downarrow_* err$ if $r \rightarrow^* r'$ and $r' \downarrow err$. Note that configurations with errors are closed under reduction. In general, we are interested in deciding whether $r \downarrow_* err$, that is, whether r can reach a configuration with error.

The process calculus presented differs from the π-calculus in two main respects: (i) We let all communications go through a unique (unnamed) channel that connects the principals to the environment. (ii) We add cryptographic primitives, which affect the contents of the cryptographic table.

In principle, we can code this process calculus in a variety of π-calculus. This amounts to: (i) Decorating all input-output actions with fresh global channels — thus replacing $!b.p$ with $\overline{a}b.p$ and likewise $?b.p$ with $a(b).p$, where a is a fresh name. (ii) Representing the cryptographic table as a process that receives messages on a global channel, say c. The coding and decoding operations are represented as remote procedure calls from the principals and the environment to the cryptographic process. To make sure that messages are not intercepted, we assume that the cryptographic process is the *unique receiver* on the channel c (syntactic conditions that guarantee this property are presented in [3]).

We refrain from going into this development because it seems much more effective, both in the mathematical development and in the practical applications, to expose directly the structure of the cryptographic table rather than hiding it behind a process algebraic veil.

Finally, we remark that the reduction rules in figure 1 can be easily turned into a labelled transition system whose actions (internal reduction, input, free and bound output) are inherited from the π-calculus. We can then rely on the π-calculus notion of bisimulation to reason about the equivalence of configurations. Thus our approach does not preclude the expression of security properties based on process equivalence [1].

Some syntactic sugar We now describe how several concepts may be encoded in the core calculus given above. We first describe several abbreviations that improve readability. We then describe annotations with which we decorate the protocols when analyzing their secrecy and integrity properties. We sometimes use *multiple* abbreviations if the order of expanding them out is apparent from the context.

Sending or receiving a tuple of names can be encoded in the calculus with monadic communication by considering tuples as ciphertexts encoded with a distinguished globally-known key g:

$$!(\vec{b}).p \equiv \mathsf{let}\ a = \{\vec{b}\}_g\ \mathsf{in}\ !a.p \qquad ?(\vec{b}).p \equiv ?a.\mathsf{case}\ \{\vec{b}\}_g = a\ \mathsf{in}\ p\ .$$

A ciphertext $\{\vec{b}\}_c$ may appear as a component of the output tuple, or as a value in a match, in an encoding or as a parameter. In all these cases it is intended that $\{\vec{b}\}_c$ stands for the name a resulting from the encoding $\mathsf{let}\ a = \{\vec{b}\}_c\ \mathsf{in}\ \ldots$ For instance:

$$\begin{array}{ll}
!(\vec{b'}, \{\vec{b}\}_c, \ldots).p & \equiv \mathsf{let}\ b = \{\vec{b}\}_c\ \mathsf{in}\ !(\vec{b'}, b, \ldots).p \\
[a = \{\vec{b}\}_c]p, q & \equiv \mathsf{let}\ b = \{\vec{b}\}_c\ \mathsf{in}\ [a = b]p, q \\
\mathsf{let}\ b = \{\vec{b'}, \{\vec{b}\}_c, \ldots\}_{c'}\ \mathsf{in}\ p & \equiv \mathsf{let}\ a = \{\vec{b}\}_c\ \mathsf{in}\ \mathsf{let}\ b = \{\vec{b'}, a, \ldots\}_{c'}\ \mathsf{in}\ p \\
A(\vec{a'}, \{\vec{b}\}_c, \ldots) & \equiv \mathsf{let}\ a = \{\vec{b}\}_c\ \mathsf{in}\ A(\vec{a'}, a, \ldots)\ .
\end{array}$$

In a filtered input, we check that the input has a component that is equal to a certain value (marked as $\underline{\underline{b}}$) or that has a certain shape, *e.g.* $\{\vec{b}\}_c$, and we stop otherwise.

$$\begin{array}{ll}
?(\vec{b'}, \underline{\underline{b}}, \ldots).p & \equiv ?(\vec{b'}, c, \ldots).[c = b]p, \mathbf{0} \\
?(\vec{b'}, \{\vec{b}\}_c, \ldots).p & \equiv ?(\vec{b'}, b, \ldots).\mathsf{case}\ \{\vec{b}\}_c = b\ \mathsf{in}\ p\ .
\end{array}$$

As in the filtered input, we check that the decryption of the ciphertext yields a certain component.

$$\begin{array}{ll}
\mathsf{case}\ \{\vec{b'}, \underline{\underline{b}}, \ldots\}_c = a\ \mathsf{in}\ p & \equiv \mathsf{case}\ \{\vec{b'}, x, \ldots\}_c = a\ \mathsf{in}\ [x = b]p, \mathbf{0} \\
\mathsf{case}\ \{\vec{b'}, \{\vec{c}\}_d, \ldots\}_{c'} = a'\ \mathsf{in}\ p & \equiv \mathsf{case}\ \{\vec{b'}, a, \ldots\}_{c'} = a'\ \mathsf{in}\ \mathsf{case}\ \{\vec{c}\}_d = a\ \mathsf{in}\ p\ .
\end{array}$$

We mark the generation of a name a intended to remain secret in a protocol configuration with the annotation $(\nu a)^{sec}\, p$. We can easily program an observer $W(a)$ such that if the environment ever discovers the name a, then an error configuration is reachable:

$$W(a) \equiv ?a'[a = a']err, \mathbf{0} \quad \text{(secrecy observer)}. \tag{1}$$

Secrecy annotations are then translated as follows:

$$(\nu a)^{sec}\, p \equiv (\nu a)\,(p \mid W(a)) \quad \text{(secrecy annotation)}. \tag{2}$$

In order to program an observer for authenticity properties we need a limited form of private channel. Fortunately, a private key a already allows the encoding of a private use-at-most-once channel a as follows:

$$\overline{a}\vec{b}.p \equiv !\{\vec{b}\}_a.p \qquad a(\vec{b}).p \equiv ?\{\vec{b}\}_a.p\ .$$

We note that this encoding does not work for arbitrary (multiple-use) channels, since the environment may replay messages, and without mechanisms like time-stamping, it would

not be possible to differentiate replayed messages from genuine fresh messages. However, the encoding can be generalized to a bounded use channel, if each message instance contains distinguished components that uniquely index each distinct use of the channel.

To specify authenticity properties we mark certain send and receive actions in the principals with authenticity annotations. The idea is as follows: let us assume that after executing its part of the protocol, principal y receives a message (f', t', m'), where f', t' name the purported sender and receiver of the message. Suppose y believes that this message is authentic, *i.e.*, has been sent to it (thus, $t' = y$) by the principal x whom it has presumably authenticated. An all-knowing 'session' judge can detect an authenticity flaw if this particular message had not actually been sent by x. The judge only rules on messages purportedly sent by x — if y presumes to have authenticated another principal, the judge reports no error. Following this intuition, we introduce authenticity annotations $auth\_o(\_), auth\_i(\_)$ which represent, respectively, x registering a message with a judge process prior to sending a message to y, and y claiming authenticity of the message received.

$$\begin{array}{ll} auth\_o((x,y,m)).p & \equiv (\nu n)\,\overline{j}(\mathsf{o},(x,y,m),n).j'(\underline{\underline{n}}).p \\ auth\_i((x,y,m)).p & \equiv \overline{j}(\mathsf{i},(x,y,m),\_).p \\ J_{x,y} & \equiv j(d,(\underline{\underline{x}},t,m),n)[d=\mathsf{o}](\overline{j'}n.J'_{x,y}(m)),([d=\mathsf{i}](err,\mathbf{0})) \\ J'_{x,y}(m) & \equiv j(d,(\underline{\underline{x}},t',m'),\_).[d=\mathsf{i}]([m'=m]\mathbf{0},err),J'_{x,y}(m)\ . \end{array}$$

Here we assume two distinguished names o and i, which indicate whether the authenticity of a message is being registered or claimed. Communication with the judge is over restricted channels j, j' to disallow the environment from making bogus assertions of authenticity. We will require the channel j appears in the principals to which x, y are instanced, and j' only in the principal corresponding to x. Note that although j is used twice, the different uses are distinguishable by the names o and i, and so replays can be recognized.

Relying on these encodings we translate authenticity annotations as follows:

$$!(\vec{b})^{auth}.p \equiv auth\_o(\vec{b}).!(\vec{b}).p \qquad ?(\vec{b})^{auth}.p \equiv ?(\vec{b}).auth\_i(\vec{b}).p\ .$$

We observe that in an authenticated output, the principal receives an acknowledgement from the judge process on channel j' *before* actually outputting the message.

3 Modelling cryptographic protocols

We now illustrate how a protocol specified informally in a notation common in the security community can be transformed into an annotated process in the enriched notation of §2. We consider a symmetric key protocol due to Yahalom. This protocol concerns principals a, b, and a trusted server c running in a possibly hostile environment that can intercept all communications. Initially, principal a and principal b each share a symmetric secret key with c. At the end of the protocol, principals a, b (and c) share a third symmetric secret key, a 'session key', which can be used by principals a and b to exchange information. A

particular run of the protocol is informally described by the following list of events:

$$
\begin{array}{lll}
\text{event } 1 & a \to b: & a, b,\ n_a \\
\text{event } 2 & b \to c: & b,\ \{a, n_a, n_b\}_{k_{bc}} \\
\text{event } 3 & c \to a: & a, \{b, k_{ab}, n_a, n_b\}_{k_{ac}},\ \{a, k_{ab}\}_{k_{bc}} \\
\text{event } 4 & a \to b: & b, \{a, k_{ab}\}_{k_{bc}},\ \{n_b\}_{k_{ab}} \\
 & & \\
\text{event } 5 & a \to b: & a, b, \{d\}_{k_{ab}} \quad post-protocol \\
\text{event } 5' & b \to a: & b, a, \{d\}_{k_{ab}} \quad alt-post-protocol\ .
\end{array}
$$

Principal a sends a clear-text message containing a nonce challenge to b. Instead of responding directly to a, b generates a new nonce n_b, its response to a's challenge, which, added to components of the original message, is sent encrypted to c. Server c creates a secret key k_{ab}, which is placed in two separately encrypted message pieces that are sent to a: the first part, readable by a, contains a's original challenge, b's retort n_b and the shared secret k_{ab}. The other part, not readable by a but by b, contains the same shared secret and a's identity; it is forwarded by a in event 4 to b, together with b's challenge *encrypted with this new shared secret*. For parametricity in the modelling, we have explicitly added redundant information in some of the messages — b in event 1, a in event 3, and b in event 4. We have also shown (two possibilities of) the first post-protocol message in which datum d is sent encrypted using the new session key k_{ab}.

The *secrecy property* we would like to verify is that the keys k_{ac}, k_{bc}, and k_{ab} remain secret. The *authenticity property* that we would like to verify is that the message successfully received by b at the second part of event 5 (alternatively $5'$) of the protocol is 'authentic', *i.e.*, it is equal to the message emitted by a in the first part of the same event. Following *event* 3, principal a will believe it is interacting with principal b if the third element of the message component encrypted with k_{ac} is the same as the nonce n_a that it had generated in *event* 1. Following *event* 4, principal b will believe it has authenticated and established a secure channel with principal a, provided the nonce n_b encrypted with the received key k_{ab} is equal to the nonce generated at *event* 2.

We will model a system *q(a,b,c)* consisting of principals a, b and c, which are assumed to follow the protocol honestly. Our goal is to verify that a session between a and b cannot be attacked by showing that the system *q(a,b,c)* can never evolve into a configuration with error.

From the message sequence chart above we can extract the sequence of events where a principal acts as a sender or a receiver, as well as the name generation, cryptographic and matching operations that it performs.

Let p_a, p_b, and p_c be the processes corresponding to instances of principals a,b,c involved in one protocol session. The process p_a, for the first alternative of the first post-protocol

event, is then specified as follows:

$$\begin{array}{ll} \text{event 1} & (\nu n_a)\,(!(a, b,\ n_a).\\ \text{event 3} & ?(\underline{a}, \{\underline{b}, k, \underline{\underline{n_a}}, n\}_{k_{ac}}, y).\\ \text{event 4} & !(b, y, \{n\}_k).\\ \text{event 5} & !(a, b, \{d\}_k)^{auth}.\mathbf{0})\ . \end{array}$$

The behaviours associated with principals b and c are defined in a similar way.

$$\begin{array}{rcl} p_b & = & ?(a', \underline{b}, n').\\ & & (\nu n_b)\,(!(b, \{a', n', n_b\}_{k_{bc}}).\\ & & ?(\underline{b}, \{\underline{\underline{a'}}, k'\}_{k_{bc}}, x').\\ & & \mathsf{case}\ \{\underline{\underline{n_b}}\}_{k'} = x'\ \mathsf{in}\\ & & ?(\underline{\underline{a'}}, \underline{b}, \{z\}_{k'})^{auth}.\mathbf{0})\ . \end{array} \qquad \begin{array}{rl} p_c & =?(\underline{b}, \{\underline{\underline{a}}, n_1, n_2\}_{k_{bc}}).\\ & (\nu k_{ab})^{sec}\,!(a, \{b, k_{ab}, n_1, n_2\}_{k_{ac}}, \{a, k_{ab}\}_{k_{bc}}).\mathbf{0}\ . \end{array}$$

The process we consider for analysis is:

$$q(a, b, c) \equiv (\nu k_{ac})^{sec}\,(\nu k_{bc})^{sec}\,(\nu j)\,(\nu j')\,(p_a \mid p_b \mid p_c \mid J_{a,b} \mid \emptyset)$$

where $\emptyset$ is the empty cryptographic table. Keys k_{ac} and k_{bc} being long-term secrets, we restrict these names at the top-level. To indicate that the authenticity judge is observing a session involving a and b, we place a judge process $J_{a,b}$ in parallel with the principals and restrict the names j, j'. What we have presented is only illustrative: we also have to consider a similar process with event $5'$ instead of event 5, and with the judge $J_{b,a}$. In general, to consider the protocol operating in more complicated scenarios, we can emend q by enriching the contexts in which the principals are placed.

4 Reachability

In the previous sections, we expressed secrecy and authenticity properties as reachability properties. We now present some results on the problem of determining whether a configuration can reach one with error.

Definition 4.1 (substitution) *A name substitution σ is a function on names that is the identity almost everywhere.*

Definition 4.2 (injective renaming) *Let r and r' be configurations. We write $r \cong r'$ if there is an injective substitution σ such that $\sigma r \equiv r'$.*

We study reachability modulo injective renaming.

Lemma 4.3 (1) *The relation $\cong$ is reflexive, symmetric, and transitive.*

(2) *If $r \cong r'$ then $r \downarrow err$ iff $r' \downarrow err$.*

(3) *If $r \cong r'$ and $r \rightarrow_R r_1$ then $\exists\, r'_1$ $r' \rightarrow_R r'_1$ and $r_1 \cong r'_1$.*

We consider rewriting modulo injective renaming.

Lemma 4.4 *Let r be a configuration. Then the set $\{r' \mid r \rightarrow_R r'\}$ where R is not (let_e^2) is finite modulo injective renaming.*

Lemma 4.5 *Let $r \equiv (\nu\{\mathbf{a}\})\,(p \mid T)$ be a configuration. Given a sort s, the set of configurations to which r may reduce by (let_e^2), while introducing a name of sort s in the cryptographic table, is finite modulo injective renaming.*

Therefore, if we can bound the sorts in the cryptographic table then the reduction relation defined in figure 1 is finitely branching modulo injective renaming.

A second important result is that all reduction rules except input are strongly confluent modulo injective renaming.

Lemma 4.6 *Let r be a configuration and suppose $r \rightarrow_{R_1} r_1$ and $r \rightarrow_{R_2} r_2$ where R_1 is not the input rule. Then*

$$\exists\, r_1', r_2' \;\; (r_1 \rightarrow_{R_1'}^{0,1} r_1', r_2 \rightarrow_{R_2'}^{0,1} r_2', \;\; \textit{and}\; r_1' \cong r_2')$$

where R_2' in not the input rule and $\rightarrow_R^{0,1}$ indicates reduction in 0 or 1 steps.

The rule (let_e^2) can be postponed except in certain particular cases.

Lemma 4.7 *Suppose r is a configuration and*

$$r \equiv (\nu\mathbf{a})\,(p \mid T) \rightarrow_{let_e^2} r' \equiv (\nu\mathbf{a})\,(p \mid T \cup \{\{\mathbf{b}\}_c = a'\}) \rightarrow_R r''$$

where $\{\mathbf{b}\}_c = a'$ is the tuple introduced by the first reduction. Then in the following cases the first reduction (let_e^2) can be postponed or eliminated:

(1) *If $R = (in)$ and the name taken in input is not a' then*

$$\exists\, r_1, r_2 \;\; (r \rightarrow_{in} r_1 \rightarrow_{let_e^2} r_2) \;\textit{and}\; r'' \cong r_2 \;.$$

(2) *If $R = (let_e^2)$ and $r'' \equiv (\nu\mathbf{a})\,(p \mid T \cup \{\{\mathbf{b}\}_c = a', \{\mathbf{b_1}\}_{c_1} = a_1'\})$ where $\{\mathbf{b_1}\}_{c_1} = a_1'$ is the tuple introduced by the second reduction and this tuple does not depend on a',* i.e., *$a' \notin \{\mathbf{b_1}, c_1\}$. Then the two (let_e^2) reductions can be permuted:*

$$r \rightarrow (\nu\mathbf{a})\,(p \mid T \cup \{\{\mathbf{b_1}\}_{c_1} = a_1'\}) \rightarrow r'' \;.$$

(3) *If $R = (let_1)$ and we have*

$$\begin{array}{ll} r' & \equiv (\nu\mathbf{a})\,(p'' \mid \mathsf{let}\ a = \{\mathbf{b}\}_c\ \mathsf{in}\ p' \mid T \cup \{\{\mathbf{b}\}_c = a'\}) \rightarrow_{let_1} \\ r'' & \equiv (\nu\mathbf{a})\,(p'' \mid [a'/a]p' \mid T \cup \{\{\mathbf{b}\}_c = a'\}) \;. \end{array}$$

Then the (let_e^2) reduction can be eliminated as follows:

$$r \rightarrow_{let^2} (\nu\mathbf{a}, a')\,([a'/a]p' \mid p'' \mid T \cup \{\{\mathbf{b}\}_c = a'\}) \rightarrow_{let_e^1} r''$$

(4) *In all other cases: $\exists\, r_1, r_2 \;\; (r \rightarrow_R r_1 \rightarrow_{let_e^2} r_2)$ and $r'' \cong r_2$.*

Next we introduce a measure $d(s)$ of a sort s which will provide an upper bound on the number of (let_e^2) reductions that might be needed for the synthesis of a name.

Definition 4.8 (sort measure) *We define a measure d on sorts as follows:*

$$d(0) = 0 \quad d(s_1, \ldots, s_n) = 1 + d(s_1) + \ldots + d(s_n) \ .$$

Lemma 4.9 *Suppose $r_1 \rightarrow_{let_e^2} \cdots \rightarrow_{let_e^2} r_{n+1} \rightarrow_{in} r_{n+2}$ where the name received in input in the last reduction has sort s and $n \geq d(s)$. Then at least $n - d(s)$ (let_e^2) reductions can be postponed modulo injective renaming,* i.e.

$$r_1 \equiv r'_1 \rightarrow_{let_e^2} \cdots \rightarrow_{let_e^2} r'_{d(s)+1} \rightarrow_{in} r'_{d(s)+2} \rightarrow_{let_e^2} \cdots \rightarrow_{let_e^2} r'_{n+2}$$

and $r'_{n+2} \cong r_{n+2}$.

PROOFHINT. We apply lemma 4.7(1) to shift the input reduction to the left till the point where the (let_e^2) reduction introduces a tuple $\{\mathbf{b}\}_c = a$ and a is the name of sort s taken in input. The construction of the name a needs at most $d(s)$ (let_e^2) reductions. All the other (let_e^2) reductions can be moved to the right of the input by iterated application of the lemma 4.7(1-2). QED

To summarize, (let_e^2) reductions can be postponed except when they are needed in the construction of a name to be input. In this case, the number of needed (let_e^2) reductions is bounded by $d(s)$ if s is the sort of the input name.

Next, let us concentrate on the reachability problem in the case where all principals are finite processes (in practical applications this is often the case). We note that the secrecy and authenticity annotations compile to finite processes.

Definition 4.10 (configuration measure) *We define the measure of a configuration $r \equiv (\nu\{\mathbf{a}\})\,(p \mid T)$ as the pair $(|p|, |\mathbf{a}|)$ where $|p|$ is the size of the process and $|\mathbf{a}|$ is the cardinality of $\{\mathbf{a}\}$.*

Lemma 4.11 (1) *Rules (out), (in), (ν), (m), (let^1), (let^2), and $(case)$ decrease the size of the process $|p|$.*

(2) *Rules (let_e^1) and $(case_e)$ decrease the size of the restricted names $|\mathbf{a}|$ while leaving unchanged the size of the processes.*

(3) *Rule (let_e^2) leaves the measure $(|p|, |\mathbf{a}|)$ unchanged.*

(4) *If $r \rightarrow_{let_e^2} r'$ and $r' \downarrow err$ then $r \downarrow err$.*

In the following, we concentrate on the issue of deciding reachability of a configuration with error assuming that for every input we can compute a *finite and complete* set of sorts which is defined as follows.

1. Perform in an arbitrary order the reductions different from (let_e^2) and (in).
2. Analyse the current configuration:
   (a) *err* has been reached: stop and report that *err* is reachable.
   (b) *err* has not been reached and no (in) reductions are possible: backtrack if possible, otherwise report that *err* is not reachable and stop.
   (c) Otherwise:
      i. Non-deterministically select an input action and compute a finite complete set of sorts for it, say $\{s_1, \ldots, s_n\}$.
      ii. Non-deterministically perform a sequence of (let_e^2) reductions of length at most $max\{d(s_1), \ldots, d(s_n)\}$.
      iii. Non-deterministically select an input for the input action. Goto step 1.

Figure 2: Complete strategy for checking error reachability

Definition 4.12 (complete set of sorts) *Given a configuration* $r \equiv (\nu \mathbf{a})\,(?a.p \mid q \mid T)$ *we say that a set of sorts* S *is* complete *for the input* $?a.p$ *if whenever there is a reduction sequence starting from* r *leading to err and whose first reduction is performed by* $?a.p$*, there is a reduction sequence leading to err where the name taken in input has a sort in* S*.*

Example 4.13 *The problem of determining tight bounds on a complete set of sorts is not trivial. Consider* $p \equiv (\nu k)\,(?a.!\{a\}_k \mid ?\{\{c\}_{k'}\}_k err)$*. It is easily checked that the set* $\{0\}$ *is not complete for the input* $?a.!\{a\}_k$*, but the set* $\{(0,0)\}$ *is.*

If a finite complete set of sorts can be computed then the strategy in figure 2 decides if r can reach *err*. We remark that sort constraints are usually assumed in the verification methods described, *e.g.*, in [11, 14].

Theorem 4.14 *Starting from a configuration* r *the strategy in figure 2 terminates and it will report an error iff* $r \downarrow_* err$*.*

PROOFHINT. Concerning termination, we note that we can perform the loop from step 1 to step $2(c)(iii)$ a finite number of times since at every iteration we perform at least one input action and this decreases the well-founded measure of definition 4.10. We will argue next that the non-deterministic choices in steps *(i)*, *(ii)*, and *(iii)* are finitely branching (modulo injective renaming). This entails the termination of the strategy.

($\Rightarrow$) The strategy examines a subset of the reachable configurations and therefore it is obviously sound.

($\Leftarrow$) Let r be the initial configuration. The rewriting in step 1 terminates in a configuration r' by lemma 4.11. By iterated application of lemma 4.6 and lemma 4.3, if $r \downarrow_* err$ then $r' \downarrow_* err$.

In step $2(b)$, if we have not reached *err* then we can safely claim by lemma 4.11(4) that *err* is not reachable.

In step $2(c)(ii)$, we know, by lemma 4.9, that if there is a sequence that leads to error then there is a sequence that leads to error whose initial sequence of (let_e^2) is bounded by the sorts' measure. By lemma 4.5, there is a finite number of sequences of (let_e^2) reductions of a given length (modulo injective renaming). Thus there are a finite number of cases to consider.

In step $2(c)(iii)$, we apply again lemma 4.4 to conclude that there are a finite number of cases to consider modulo injective renaming. QED

It is not difficult to generate effectively a positive boolean combination of equational constraints on sorts which determines the general shape of the input variables. As above, we consider configurations $r \equiv (\nu \mathbf{a})\,(p \mid T)$ such that p does not contain recursive definitions. Let $p_1, \ldots, p_n$ be the list of all sequential threads resulting from the interleaving of the actions in p. We remark that to determine whether $r \downarrow_* err$ it is sufficient to check whether $\exists\, i \;\; (\nu \mathbf{a})\,(p_i \mid T) \downarrow_* err$. Therefore, without loss of generality, we consider processes that do not contain parallel composition. We also assume that all bound names in r are distinct and different from the free names. To every name a we associate a distinct sort variable s_a ranging over Srt. We define a function E from configurations to constraints as follows:

$$\begin{array}{rcl} E(r) & = & \bigwedge\{s_a = srt_T(a) \mid a \in n(T)\} \wedge \\ & & \bigwedge\{s_a = 0 \mid a \in (fn(r) \cup \{\mathbf{a}\}) \backslash n(T)\} \wedge \\ & & E(p) \end{array}$$

where $E(p)$ is inductively defined on the structure of the sequential thread p as follows:

$$\begin{array}{ll} E(\mathbf{0}) = false & E([a=b]p, q) = ((s_a = s_b) \wedge E(p)) \vee E(q) \\ E(err) = true & E((\nu a)\, p) = (s_a = 0) \wedge E(p) \\ E(!a.p) = E(p) & E(\mathsf{let}\ a = \{\mathbf{b}\}_c\ \mathsf{in}\ p) = (s_a = (\mathbf{s_b}, s_c)) \wedge E(p) \\ E(?a.p) = E(p) & E(\mathsf{case}\ \{\mathbf{b}\}_c = a\ \mathsf{in}\ p) = (s_a = (\mathbf{s_b}, s_c)) \wedge E(p) \end{array}$$

With every run which reaches err, say $r \rightarrow^* r' \downarrow err$ we associate an assignment $\rho \equiv (s_{c_1} = s_1) \wedge \cdots \wedge (s_{c_n} = s_n)$ where $c_1, \ldots, c_n$ are the input variables which are actually instantiated in the run and $s_1, \ldots, s_n$ are the corresponding sorts of the names provided by the environment. The assignment ρ satisfies the constraints $E(r)$.

Proposition 4.15 *If $r \rightarrow^* r' \downarrow err$ and ρ is the corresponding assignment of input variables then $E(r) \wedge \rho$ is consistent.*

PROOFHINT. By induction on the length of the reduction $r \rightarrow^* r'$. QED

For instance, in example 4.13, we can deduce that the sort of the input on the right hand side must have the shape $((s, 0), 0)$. Of course, there are infinitely many sorts of this shape and therefore proposition 4.15 does not provide a *finite* complete set of sorts. It remains to be seen whether *symbolic* representations of the configurations can yield stronger decidability results.

Acknowledgements. We would like to thank Kevin Compton, Mads Dam, Andrew Gordon, and Joachim Parrow for helpful discussions.

References

[1] M. Abadi and A. Gordon. A calculus for cryptographic protocols: the spi calculus. In *Proc. ACM Computer and Comm. Security*, 1997.

[2] M. Abadi and A. Gordon. A bisimulation method for cryptographic protocols. In *Proc. ESOP 98, Springer Lect. Notes in Comp. Sci. 1382*, 1998.

[3] R. Amadio. On modeling mobility. *Journal of Theoretical Computer Science, to appear*, 1999.

[4] D. Bolignano. Formal verification of cryptographic protocols. In *Proc. ACM Conference on Computer Communication and Security*, 1996.

[5] M. Boreale, R. De Nicola, and R. Pugliese. Proof techniques for cryptographic processes. In *Proc. IEEE Logic in Comp. Sci.*, 1999.

[6] E. Clarke, S. Jha, and W. Marrero. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In *Proc. IFIP Working Conference on Programming Concepts and Methods (PROCOMET)*, 1998.

[7] M. Dam. Proving trust in systems of second-order processes: preliminary results. In *Proc. HICSS 98*, 1998.

[8] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Trans. on Information Theory*, 29(2):198–208, 1983.

[9] M. Elkjaer, M. Höle, H. Hüttel, and K. Nielsen. Towards automatic bisimilarity checking in the Spi calculus. In *Proc. of DMTCS 99 and CATS 99*, 1999.

[10] A. Huima. Efficient infinite-state analysis of security protocols. In *Proc. Formal methods and security protocols, FLOC Workshop, Trento*, 1999.

[11] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using fdr. In *Proc. TACAS, Springer Lect. Notes in Comp. Sci. 1996*, 1996.

[12] C. Meadows. A model of computation for the nrl protocol analyzer. In *Proc. IEEE Computer Security Foundations Workshop*, 1994.

[13] J. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murϕ. In *Proc. IEEE Symposium on Security and Privacy*, 1997.

[14] L. Paulson. Proving properties of security protocols by induction. In *Proc. IEEE Computer Security Foundations Workshop*, 1997.

[15] S. Schneider. Security properties and CSP. In *Proc. IEEE Symp. Security and Privacy*, 1996.

[16] C. Weidenbach. Towards an automatic analysis of security protocols in first-order logic. In *Proc. CADE 99*. Springer Lect. Notes in Comp. Sci. (LNAI) 1632, 1999.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - B.P. 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Lorraine : Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 Villers lès Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot St Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, B.P. 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399